Operation codes

G.SET.AND.E.8	Group set and equal zero bytes
G.SET.AND.E.16	Group set and equal zero doublets
G.SET.AND.E.32	Group set and equal zero quadlets
G.SET.AND.E.64	Group set and equal zero octlets
G.SET.AND.E.128	Group set and equal zero hexlet
G.SET.AND.NE.8	Group set and not equal zero bytes
G.SET.AND.NE.16	Group set and not equal zero doublets
G.SET.AND.NE.32	Group set and not equal zero quadlets
G.SET.AND.NE.64	Group set and not equal zero octlets
G.SET.AND.NE.128	Group set and not equal zero hexlet
G.SET.E.8	Group set equal bytes
G.SET.E.16	Group set equal doublets
G.SET.E.32	Group set equal quadlets
G.SET.E.64	Group set equal octlets
G.SET.E.128	Group set equal hexlet
G.SET.GE.8	Group set greater equal signed bytes
G.SET.GE.16	Group set greater equal signed doublets
G.SET.GE.32	Group set greater equal signed quadlets
G.SET.GE.64	Group set greater equal signed octlets
G.SET.GE.128	Group set greater equal signed hexlet
G.SET.GE.U.8	Group set greater equal unsigned bytes
G.SET.GE.U.16	Group set greater equal unsigned doublets
G.SET.GE.U.32	Group set greater equal unsigned quadlets
G.SET.GE.U.64	Group set greater equal unsigned octlets
G.SET.GE.U.128	Group set greater equal unsigned hexlet
G.SET.L.8	Group set signed less bytes
G.SET.L.16	Group set signed less doublets
G.SET.L.32	Group set signed less quadlets
G.SET.L.64	Group set signed less octlets
G.SET.L.128	Group set signed less hexlet
G.SET.L.U.8	Group set less unsigned bytes
G.SET.L.U.16	Group set less unsigned doublets
G.SET.L.U.32	Group set less unsigned quadlets
G.SET.L.U.64	Group set less unsigned octlets
G.SET.L.U.128	Group set less unsigned hexlet
G.SET.NE.8	Group set not equal bytes
G.SET.NE.16	Group set not equal doublets

Fig. 33A

G.SET.NE.32	Group set not equal quadlets
G.SET.NE.64	Group set not equal octlets
G.SET.NE.128	Group set not equal hexlet
G.SUB.8	Group subtract bytes
G.SUB.8.O	Group subtract signed bytes check overflow
G.SUB.16	Group subtract doublets
G.SUB.16.O	Group subtract signed doublets check overflow
G.SUB.32	Group subtract quadlets
G.SUB.32.O	Group subtract signed quadlets check overflow
G.SUB.64	Group subtract octlets
G.SUB.64.O	Group subtract signed octlets check overflow
G.SUB.128	Group subtract hexlet
G.SUB.128.O	Group subtract signed hexlet check overflow
G.SUB.L.8	Group subtract limit signed bytes
G.SUB.L.16	Group subtract limit signed doublets
G.SUB.L.32	Group subtract limit signed quadlets
G.SUB.L.64	Group subtract limit signed octlets
G.SUB.L.128	Group subtract limit signed hexlet
G.SUB.L.U.8	Group subtract limit unsigned bytes
G.SUB.L.U.16	Group subtract limit unsigned doublets
G.SUB.L.U.32	Group subtract limit unsigned quadlets
G.SUB.L.U.64	Group subtract limit unsigned octlets
G.SUB.L.U.128	Group subtract limit unsigned hexlet
G.SUB.U.8.O	Group subtract unsigned bytes check overflow
G.SUB.U.16.O	Group subtract unsigned doublets check overflow
G.SUB.U.32.O	Group subtract unsigned quadlets check overflow
G.SUB.U.64.O	Group subtract unsigned octlets check overflow
G.SUB.U.128.O	Group subtract unsigned hexlet check overflow

Fig. 33A (cont'd)

Equivalencies

G.SET.E.Z.8	Group set equal zero bytes
G.SET.E.Z.16	Group set equal zero doublets
G.SET.E.Z.32	Group set equal zero quadlets
G.SET.E.Z.64	Group set equal zero octlets
G.SET.E.Z.128	Group set equal zero hexlet
G.SET.G.Z.8	Group set greater zero signed bytes
G.SET.G.Z.16	Group set greater zero signed doublets
G.SET.G.Z.32	Group set greater zero signed quadlets
G.SET.G.Z.64	Group set greater zero signed octlets
G.SET.G.Z.128	Group set greater zero signed hexlet
G.SET.GE.Z.8	Group set greater equal zero signed bytes
G.SET.GE.Z.16	Group set greater equal zero signed doublets
G.SET.GE.Z.32	Group set greater equal zero signed quadlets
G.SET.GE.Z.64	Group set greater equal zero signed octlets
G.SET.GE.Z.128	Group set greater equal zero signed hexlet
G.SET.L.Z.8	Group set less zero signed bytes
G.SET.L.Z.16	Group set less zero signed doublets
G.SET.L.Z.32	Group set less zero signed quadlets
G.SET.L.Z.64	Group set less zero signed octlets
G.SET.L.Z.128	Group set less zero signed hexlet
G.SET.LE.Z.8	Group set less equal zero signed bytes
G.SET.LE.Z.16	Group set less equal zero signed doublets
G.SET.LE.Z.32	Group set less equal zero signed quadlets
G.SET.LE.Z.64	Group set less equal zero signed octlets
G.SET.LE.Z.128	Group set less equal zero signed hexlet
G.SET.NE.Z.8	Group set not equal zero bytes
G.SET.NE.Z.16	Group set not equal zero doublets
G.SET.NE.Z.32	Group set not equal zero quadlets
G.SET.NE.Z.64	Group set not equal zero octlets
G.SET.NE.Z.128	Group set not equal zero hexlet

Fig. 33A (cont'd)

G.SET.LE.8	Group set less equal signed bytes
G.SET.LE.16	Group set less equal signed doublets
G.SET.LE.32	Group set less equal signed quadlets
G.SET.LE.64	Group set less equal signed octlets
G.SET.LE.128	Group set less equal signed hexlet
G.SET.LE.U.8	Group set less equal unsigned bytes
G.SET.LE.U.16	Group set less equal unsigned doublets
G.SET.LE.U.32	Group set less equal unsigned quadlets
G.SET.LE.U.64	Group set less equal unsigned octlets
G.SET.LE.U.128	Group set less equal unsigned hexlet
G.SET.G.8	Group set signed greater bytes
G.SET.G.16	Group set signed greater doublets
G.SET.G.32	Group set signed greater quadlets
G.SET.G.64	Group set signed greater octlets
G.SET.G.128	Group set signed greater hexlet
G.SET.G.U.8	Group set greater unsigned bytes
G.SET.G.U.16	Group set greater unsigned doublets
G.SET.G.U.32	Group set greater unsigned quadlets
G.SET.G.U.64	Group set greater unsigned octlets
G.SET.G.U.128	Group set greater unsigned hexlet

G.SET.E.Z.size rd=rc	←	G.SET.AND.E.size rd=rc,rc
G.SET.G.Z.size rd=rc	⇐	G.SET.L.U.size rd=rc,rc
G.SET.GE.Z.size rd=rc	⇐	G.SET.GE.size rd=rc,rc
G.SET.L.Z.size rd=rc	⇐	G.SET.L.size rd=rc,rc
G.SET.LE.Z.size rd=rc	⇐	G.SET.GE.U.size rd=rc,rc
G.SET.NE.Z.size rd=rc	←	G.SET.AND.NE.size rd=rc,rc
G.SET.G.size rd=rb,rc	→	G.SET.L.size rd=rc,rb
G.SET.G.U.size rd=rb,rc	→	G.SET.L.U.size rd=rc,rb
G.SET.LE.size rd=rb,rc	→	G.SET.GE.size rd=rc,rb
G.SET.LE.U.size rd=rb,rc	→	G.SET.GE.U.size rd=rc,rb

Fig. 33A (cont'd)

Definition

```
def mul(size,h,vs,v,i,ws,w,j) as
        mul ← ((vs&v_{size-1+i})^{h-size} || v_{size-1+i..i}) * ((ws&w_{size-1+j})^{h-size} || w_{size-1+j..j})
enddef

def c ← PolyMultiply(size,a,b) as
        p[0] ← 0^{2*size}
        for k ← 0 to size-1
                p[k+1] ← p[k] ^ a_k ? (0^{size-k} || b || 0^k) : 0^{2*size}
        endfor
        c ← p[size]
enddef

def Ensemble(op,size,rd,rc,rb)
        c ← RegRead(rc, 128)
        b ← RegRead(rb, 128)
        case op of
                E.MUL:, E.MUL.C:, EMUL.SUM, E.MUL.SUM.C, E.CON, E.CON.C, E.DIV:
                        cs ← bs ← 1
                E.MUL.M:, EMUL.SUM.M, E.CON.M:
                        cs ← 0
                        bs ← 1
                E.MUL.U:, EMUL.SUM.U, E.CON.U, E.DIV.U, E.MUL.P:
                        cs ← bs ← 0
        endcase
        case op of
                E.MUL, E.MUL.U, E.MUL.M:
                        for i ← 0 to 64-size by size
                                d_{2*(i+size)-1..2*i} ← mul(size,2*size,cs,c,i,bs,b,i)
                        endfor
                E.MUL.P:
                        for i ← 0 to 64-size by size
                                d_{2*(i+size)-1..2*i} ← PolyMultiply(size,c_{size-1+i..i},b_{size-1+i..i})
                        endfor
                E.MUL.C:
                        for i ← 0 to 64-size by size
                                if (i and size) = 0 then
                                        p ← mul(size,2*size,1,c,i,1,b,i) - mul(size,2*size,1,c,i+size,1,b,i+size)
                                else
                                        p ← mul(size,2*size,1,c,i,1,b,i+size) + mul(size,2*size,1,c,i,1,b,i+size)
                                endif
                                d_{2*(i+size)-1..2*i} ← p
                        endfor
                E.MUL.SUM, E.MUL.SUM.U, E.MUL.SUM.M:
                        p[0] ← 0^{128}
                        for i ← 0 to 128-size by size
                                p[i+size] ← p[i] + mul(size,128,cs,c,i,bs,b,i)
                        endfor
                        a ← p[128]
                E.MUL.SUM.C:
                        p[0] ← 0^{64}
                        p[size] ← 0^{64}
                        for i ← 0 to 128-size by size
                                if (i and size) = 0 then
                                        p[i+2*size] ← p[i] + mul(size,64,1,c,i,1,b,i)
                                                          - mul(size,64,1,c,i+size,1,b,i+size)
                                else
                                        p[i+2*size] ← p[i] + mul(size,64,1,c,i,1,b,i+size)
                                                          + mul(size,64,1,c,i+size,1,b,i)
                                endif
                        endfor
                        a ← p[128+size] || p[128]
```

Fig. 34C

```
            E.CON, E.CON.U, E.CON.M:
                p[0] ← 0^128
                for j ← 0 to 64-size by size
                    for i ← 0 to 64-size by size
                        p[j+size]2*(i+size)-1..2*i ← p[j]2*(i+size)-1..2*i +
                            mul(size,2*size,cs,c,i+64-j,bs,b,j)
                    endfor
                endfor
                a ← p[64]
            E.CON.C:
                p[0] ← 0^128
                for j ← 0 to 64-size by size
                    for i ← 0 to 64-size by size
                        if ((~i) and j and size) = 0 then
                            p[j+size]2*(i+size)-1..2*i ← p[j]2*(i+size)-1..2*i +
                                mul(size,2*size,1,c,i+64-j,1,b,j)
                        else
                            p[j+size]2*(i+size)-1..2*i ← p[j]2*(i+size)-1..2*i -
                                mul(size,2*size,1,c,i+64-j+2*size,1,b,j)
                        endif
                    endfor
                endfor
                a ← p[64]
            E.DIV:
                if (b = 0) or ( (c = (1||0^63)) and (b = 1^64) ) then
                    a ← undefined
                else
                    q ← c / b
                    r ← c - q*b
                    a ← r63..0 || q63..0
                endif
            E.DIV.U:
                if b = 0 then
                    a ← undefined
                else
                    q ← (0 || c) / (0 || b)
                    r ← c - (0 || q)*(0 || b)
                    a ← r63..0 || q63..0
                endif
        endcase
        RegWrite(rd, 128, a)
enddef
```

Exceptions

none

Fig. 34C (cont'd)

Operation codes

E.SCAL.ADD.F.16	Ensemble scale add floating-point half
E.SCAL.ADD.F.32	Ensemble scale add floating-point single
E.SCAL.ADD.F.64	Ensemble scale add floating-point double

Fig. 38G

Selection

class	op	prec
scale add	E.SCAL.ADD.F	16 32 64

Format

E.SCAL.ADD.F.size ra=rd,rc,rb

ra=escaladdfsize(rd,rc,rb)

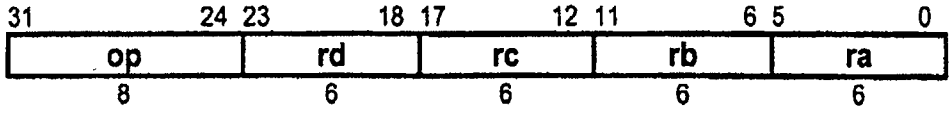


Fig. 38H

Format

E.op.prec.round rd=rc

rd=eopprecround(rc)

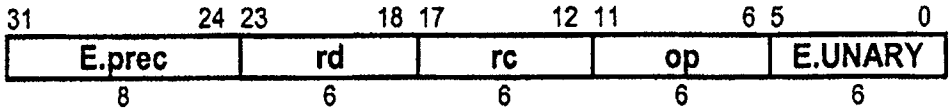


Fig. 41B

Operation codes

X.SHL.M.2	Crossbar shift left merge pecks
X.SHL.M.4	Crossbar shift left merge nibbles
X.SHL.M.8	Crossbar shift left merge bytes
X.SHL.M.16	Crossbar shift left merge doublets
X.SHL.M.32	Crossbar shift left merge quadlets
X.SHL.M.64	Crossbar shift left merge octlets
X.SHL.M.128	Crossbar shift left merge hexlet
X.SHR.M.2	Crossbar shift right merge pecks
X.SHR.M.4	Crossbar shift right merge nibbles
X.SHR.M.8	Crossbar shift right merge bytes
X.SHR.M.16	Crossbar shift right merge doublets
X.SHR.M.32	Crossbar shift right merge quadlets
X.SHR.M.64	Crossbar shift right merge octlets
X.SHR.M.128	Crossbar shift right merge hexlet

Fig. 43E

Operation codes

X.COMPRESS.I.2	Crossbar compress immediate signed pecks
X.COMPRESS.I.4	Crossbar compress immediate signed nibbles
X.COMPRESS.I.8	Crossbar compress immediate signed bytes
X.COMPRESS.I.16	Crossbar compress immediate signed doublets
X.COMPRESS.I.32	Crossbar compress immediate signed quadlets
X.COMPRESS.I.64	Crossbar compress immediate signed octlets
X.COMPRESS.I.128	Crossbar compress immediate signed hexlet
X.COMPRESS.I.U.2	Crossbar compress immediate unsigned pecks
X.COMPRESS.I.U.4	Crossbar compress immediate unsigned nibbles
X.COMPRESS.I.U.8	Crossbar compress immediate unsigned bytes
X.COMPRESS.I.U.16	Crossbar compress immediate unsigned doublets
X.COMPRESS.I.U.32	Crossbar compress immediate unsigned quadlets
X.COMPRESS.I.U.64	Crossbar compress immediate unsigned octlets
X.COMPRESS.I.U.128	Crossbar compress immediate unsigned hexlet
X.EXPAND.I.2	Crossbar expand immediate signed pecks
X.EXPAND.I.4	Crossbar expand immediate signed nibbles
X.EXPAND.I.8	Crossbar expand immediate signed bytes
X.EXPAND.I.16	Crossbar expand immediate signed doublets
X.EXPAND.I.32	Crossbar expand immediate signed quadlets
X.EXPAND.I.64	Crossbar expand immediate signed octlets
X.EXPAND.I.128	Crossbar expand immediate signed hexlet
X.EXPAND.I.U.2	Crossbar expand immediate unsigned pecks
X.EXPAND.I.U.4	Crossbar expand immediate unsigned nibbles
X.EXPAND.I.U.8	Crossbar expand immediate unsigned bytes
X.EXPAND.I.U.16	Crossbar expand immediate unsigned doublets
X.EXPAND.I.U.32	Crossbar expand immediate unsigned quadlets
X.EXPAND.I.U.64	Crossbar expand immediate unsigned octlets
X.EXPAND.I.U.128	Crossbar expand immediate unsigned hexlet
X.ROTL.I.2	Crossbar rotate left immediate pecks
X.ROTL.I.4	Crossbar rotate left immediate nibbles
X.ROTL.I.8	Crossbar rotate left immediate bytes
X.ROTL.I.16	Crossbar rotate left immediate doublets
X.ROTL.I.32	Crossbar rotate left immediate quadlets
X.ROTL.I.64	Crossbar rotate left immediate octlets
X.ROTL.I.128	Crossbar rotate left immediate hexlet
X.ROTR.I.2	Crossbar rotate right immediate pecks
X.ROTR.I.4	Crossbar rotate right immediate nibbles
X.ROTR.I.8	Crossbar rotate right immediate bytes
X.ROTR.I.16	Crossbar rotate right immediate doublets
X.ROTR.I.32	Crossbar rotate right immediate quadlets
X.ROTR.I.64	Crossbar rotate right immediate octlets
X.ROTR.I.128	Crossbar rotate right immediate hexlet

Fig. 43H

X.SHL.I.2	Crossbar shift left immediate pecks
X.SHL.I.2.O	Crossbar shift left immediate signed pecks check overflow
X.SHL.I.4	Crossbar shift left immediate nibbles
X.SHL.I.4.O	Crossbar shift left immediate signed nibbles check overflow
X.SHL.I.8	Crossbar shift left immediate bytes
X.SHL.I.8.O	Crossbar shift left immediate signed bytes check overflow
X.SHL.I.16	Crossbar shift left immediate doublets
X.SHL.I.16.O	Crossbar shift left immediate signed doublets check overflow
X.SHL.I.32	Crossbar shift left immediate quadlets
X.SHL.I.32.O	Crossbar shift left immediate signed quadlets check overflow
X.SHL.I.64	Crossbar shift left immediate octlets
X.SHL.I.64.O	Crossbar shift left immediate signed octlets check overflow
X.SHL.I.128	Crossbar shift left immediate hexlet
X.SHL.I.128.O	Crossbar shift left immediate signed hexlet check overflow
X.SHL.I.U.2.O	Crossbar shift left immediate unsigned pecks check overflow
X.SHL.I.U.4.O	Crossbar shift left immediate unsigned nibbles check overflow
X.SHL.I.U.8.O	Crossbar shift left immediate unsigned bytes check overflow
X.SHL.I.U.16.O	Crossbar shift left immediate unsigned doublets check overflow
X.SHL.I.U.32.O	Crossbar shift left immediate unsigned quadlets check overflow
X.SHL.I.U.64.O	Crossbar shift left immediate unsigned octlets check overflow
X.SHL.I.U.128.O	Crossbar shift left immediate unsigned hexlet check overflow
X.SHR.I.2	Crossbar signed shift right immediate pecks
X.SHR.I.4	Crossbar signed shift right immediate nibbles
X.SHR.I.8	Crossbar signed shift right immediate bytes
X.SHR.I.16	Crossbar signed shift right immediate doublets
X.SHR.I.32	Crossbar signed shift right immediate quadlets
X.SHR.I.64	Crossbar signed shift right immediate octlets
X.SHR.I.128	Crossbar signed shift right immediate hexlet
X.SHR.I.U.2	Crossbar shift right immediate unsigned pecks
X.SHR.I.U.4	Crossbar shift right immediate unsigned nibbles
X.SHR.I.U.8	Crossbar shift right immediate unsigned bytes
X.SHR.I.U.16	Crossbar shift right immediate unsigned doublets
X.SHR.I.U.32	Crossbar shift right immediate unsigned quadlets
X.SHR.I.U.64	Crossbar shift right immediate unsigned octlets
X.SHR.I.U.128	Crossbar shift right immediate unsigned hexlet

Fig. 43H (cont)

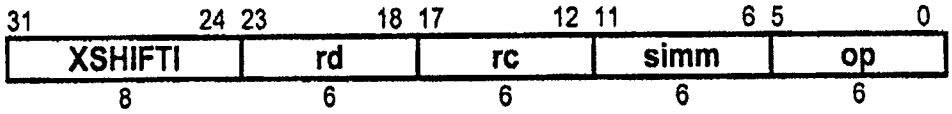
Selection

class	op	size
precision	COMPRESS.I COMPRESS.I.U EXPAND.I EXPAND.I.U	2 4 8 16 32 64 128
shift	ROTL.I ROTR.I SHL.I SHL.I.O SHL.I.U.O SHR.I SHR.I.U	2 4 8 16 32 64 128
copy	*COPY*	

Format

X.op.size rd=rc,shift

rd=xopsize(rc,shift)



$t \leftarrow 256-2*size+shift$

$op_{1..0} \leftarrow t_{7..6}$

$simm \leftarrow t_{5..0}$

Fig. 43I

Operation codes

E.MUL.X	Ensemble multiply extract
E.EXTRACT	Ensemble extract
E.SCAL.ADD.X	Ensemble scale add extract

Fig. 44E

Format

E.op ra=rd,rc,rb

ra=eop(rd,rc,rb)

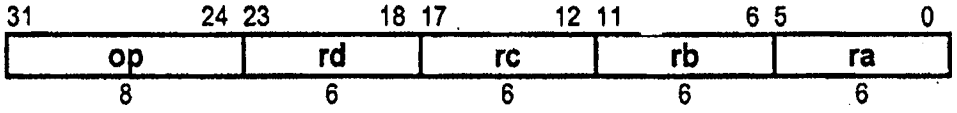


Fig. 44F

Selection

number format	type	size	alignment	ordering
signed byte		8		
unsigned byte	U	8		
signed integer		16 32 64		L B
signed integer aligned		16 32 64	A	L B
unsigned integer	U	16 32 64		L B
unsigned integer aligned	U	16 32 64	A	L B
register		128		L B
register aligned		128	A	L B

Format

op rd=rc,rb

rd=op(rc,rb)

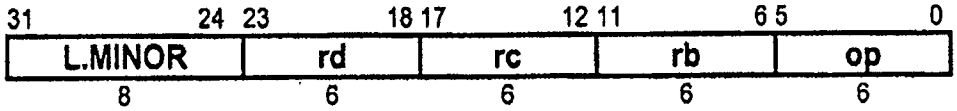


Fig. 50B

Selection

number format	op	size	alignment	ordering
byte		8		
integer		16 32 64 128		L B
integer aligned		16 32 64 128	A	L B
multiplex	MUX	64	A	L B

Format

op rd,rc,rb

op(rd,rc,rb)

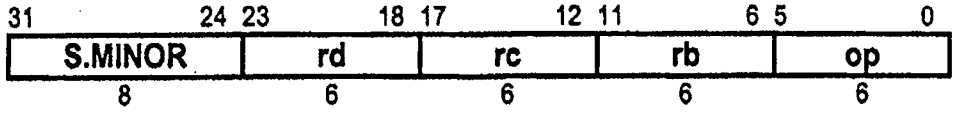


Fig. 52B